Ground Software Technologies – Embracing Change: Mission Drivers and Technology Opportunities to Enable Long Lived Missions

Brian J. Giovannoni

Jet Propulsion Laboratory / California Institute of Technology

SpaceOps May 18 - June 1 2018, Marseille, France

Agenda

- Take Away
  - Containerization and Continuous Integration can:
    - Support managing older software - **improving maintainability**
    - Allow for change (e.g. patches, OS updates, address security vulnerabilities, infrastructure) – **evolving the system**
    - Reduce testing costs in operations – **do more with less**
  - Plan for maintenance and change for the entire mission lifecycle
- Introduction
- Containerizing Software – **improving maintainability**
- Continuous Integration Approach
  - **Evolving the system**
  - **Do more with less**
- Summary

AIAA
The World's Forum for Aerospace Leadership

Introduction

- Containerization (defined)
  - A container is a stand-alone, executable image bundling software (an application) and everything needed for it to run.
- Continuous Integration (defined)
  - Continuous Integration (CI) is a derivative of agile software development practices in which developers continually check in code for a nightly build process.
  - Builds are regularly run against automated regression testing and integration problems addressed very frequently.

Introduction – Enabling Technologies

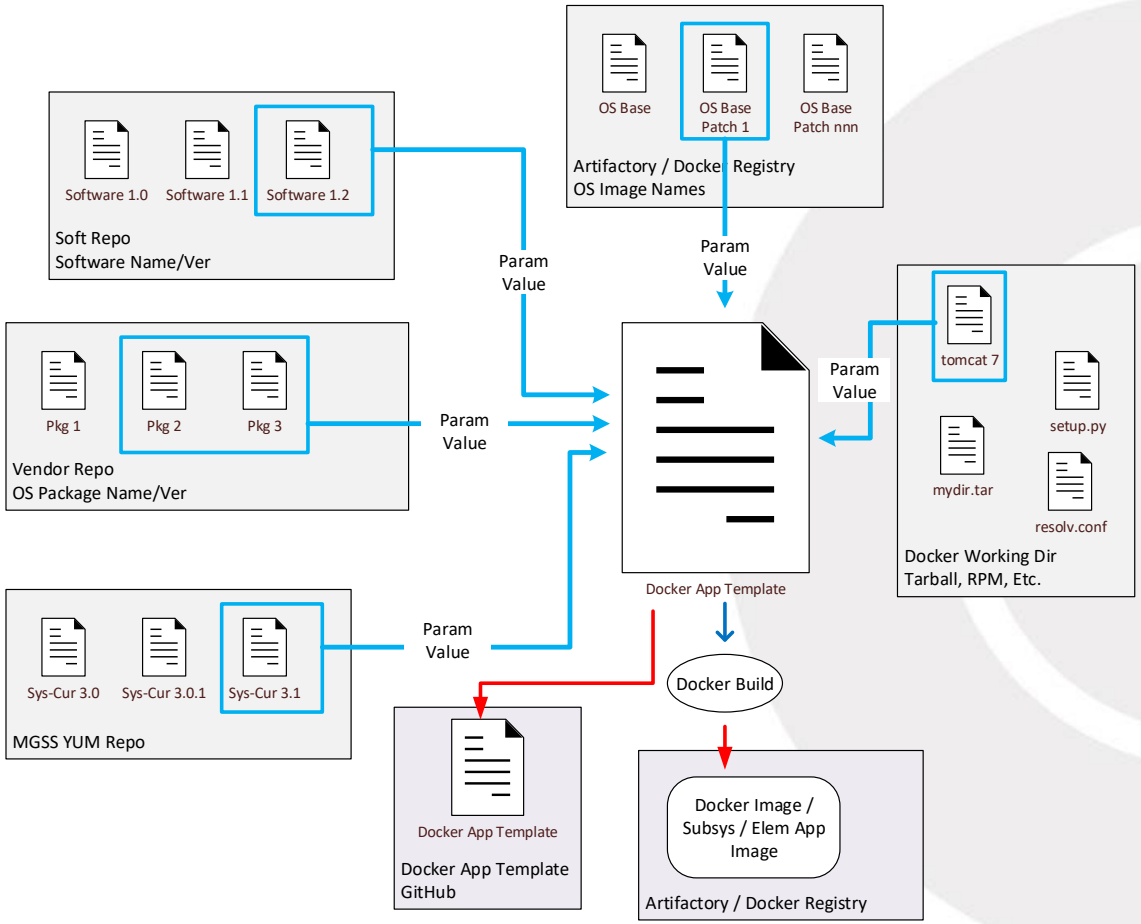
Technology	Reference
Artifactory	A development tool that supports binary management, works with different software package management systems, and easily integrates into a continuous integration workflow URL: https://jfrog.com/artifactory/
Jenkins	An open source automation server that supports building, deploying and automating development projects URL: https://jenkins.io/
Software Repositories	GIT / GITHUB SVN CVS
YUM – Yellowdog Updater Modified	A package installer/remover used for Redhat Package Managed systems URL: http://yum.baseurl.org/

Introduction – Docker (50K View & Best Practices)

- Put very simply – Docker is an application level virtual engine
  - Provides an environment for applications to execute (container)
  - Contains application binaries and system libraries need to run
  - Storage access is provided by container as needed by the application
- What types of applications are best for Docker?
  - Daemon processes
  - Pipeline processes
  - Background jobs
  - Minimal or no command line interface
- Docker is not:
  - A full virtual machine
  - Not well suited for GUI based applications
  - Not meant for real-time . Hard real-time applications

URL: https://www.docker.com/what-docker

Containerizing Software – **Improving Maintainability (1)**



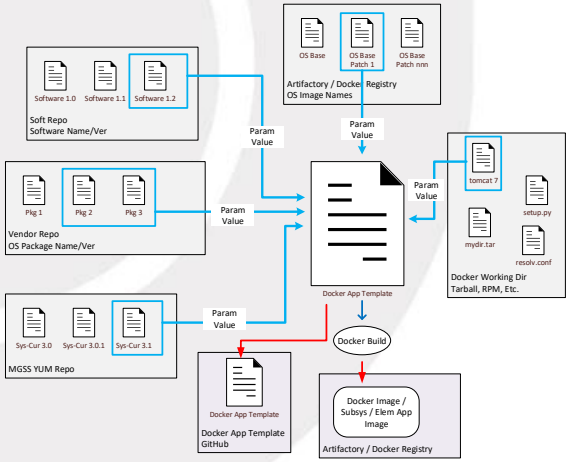
- Package only what you need – **maintainable bundles**
- Version control templates – **know what you have / back out changes**
- Software comes from sanctioned sources
- Privileged access is not required
  - Any user can instantiate a Docker image / container

Containerizing Software – **Improving Maintainability (2)**

- Templates supports
  - Defining configurable bundles
  - Versioning
- Arguments allow for:
  - Automation
  - Versioning
  - Reuse

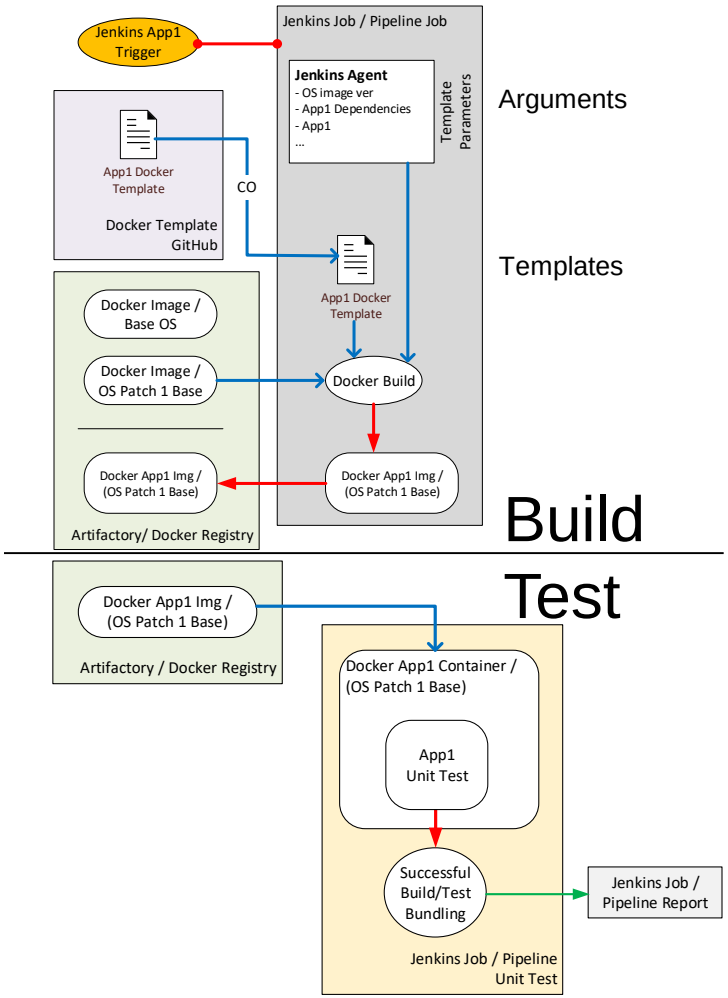
Containerizing Software – **Improving Maintainability (3)**

- Docker containers allow for:
  - Application isolation - **maintainability**
    - Runtime environment is consistent on multiple platforms
    - Isolate applications for security
  - System dependencies - **maintainability**
    - Breaks application in to modular bits containing only what is needed to execute
    - Templates document application dependencies
  - Minimal functionality
    - Containerize applications and dependencies only
    - Divide functionality in to individual processes
    - Define and understand communications path between containers
  - Templates can be versioned and used in automation workflows
  - Agnostic containers - **maintainability**
    - Bundle application and dependencies into a single object
    - Abstract machine specific settings
    - Applications can run on many different machines
    - Avoid OS, version and kernel specific dependencies and references



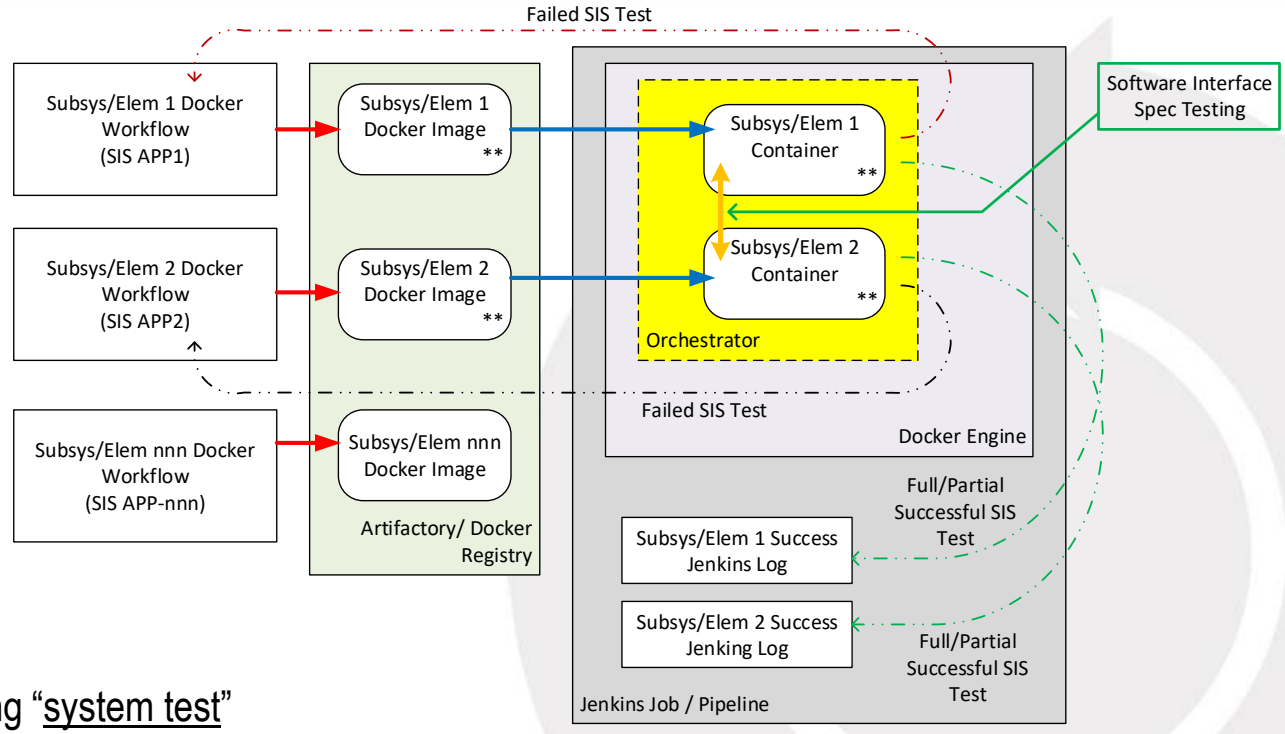
- Note: Hardest change for a legacy system is to update to allow agnostic containers

Continuous Integration Approach – **Evolving the System (1)**



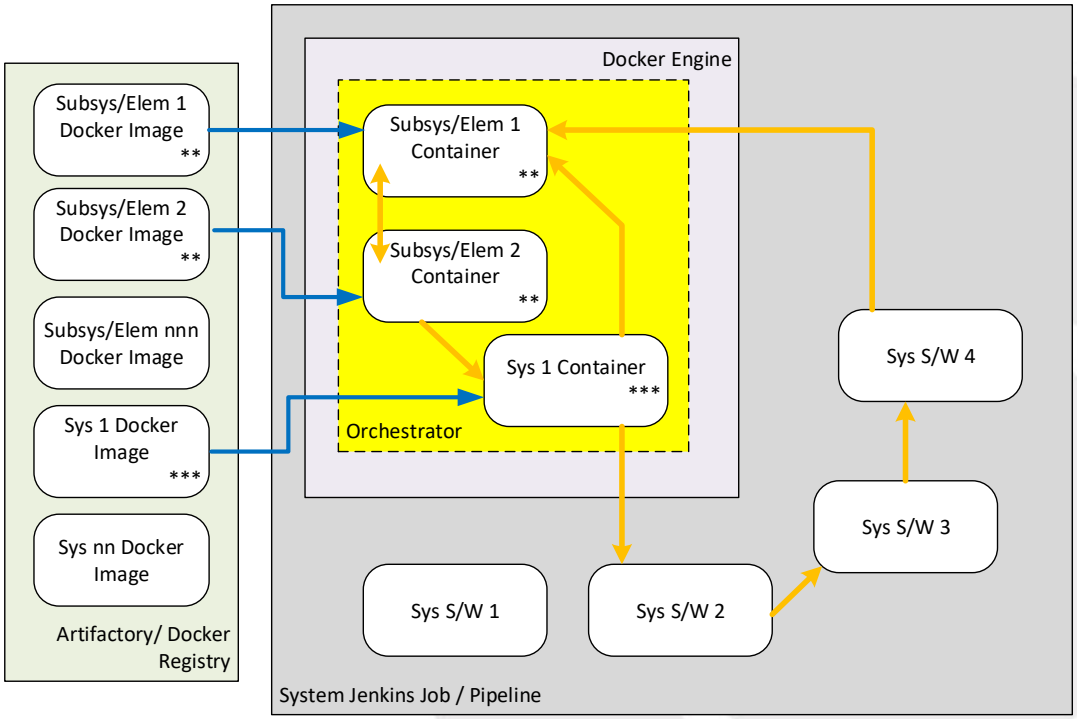
- Automate application builds
  - Daily builds reduce application integration efforts
    - Note: once teams get into a regular battle rhythm
  - Builds stored in Artifactory
    - Provides versions traceable to requirements updates
      - » Bug fixes / new requirements manages in Jira
- Automate unit tests
  - Application builds, test cases, test data and test results stored together
- Testing
  - On premise – in dedicated test servers
  - Amazon Web Services (AWS)
    - Amazon Elastic Container Service (ECS)
- Binary versioning
  - Artifactory stores application versions and test results
- **Evolving the system**
  - Applications built frequently - OS, third party software dependency patching and integration testing

Continuous Integration Approach – **Evolving the System (2)**



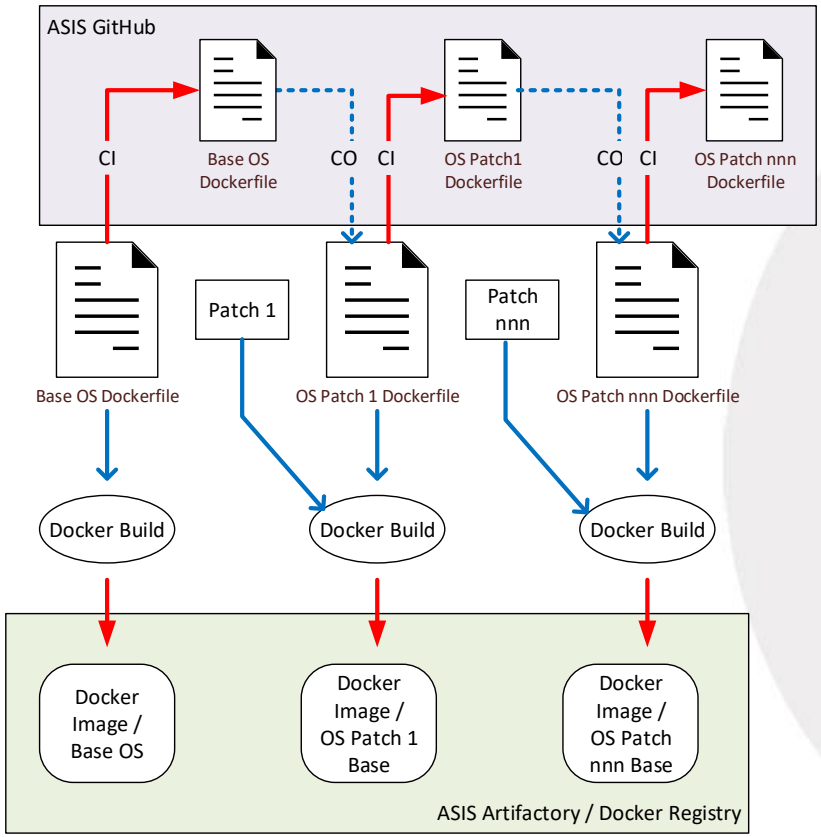
- Automating "system test"
  - Capitalize on application level build and test
    - Depending on need unit tests can be re-run
  - Test subsystem interfaces
    - Data flow
  - System thread tests
    - Performance requirements
    - Security requirements
    - System level requirements testing end-to-end

Continuous Integration Approach – **Evolving the System (3)**



- Integrate and automate containerize with non-containerized applications
- Evolving the system
  - Automated thread tests support identifying a single application update
    - Reduce the need for lengthy Integration Test & Deployment cycles – support smaller teams (especially in operations)
  - Change deployment models for containerize applications

Continuous Integration Approach – **Evolving the System (4)**



- Evolving the system
  - Patching the system (OS / 3rd Party)
    - Updates to the system can be applied and tested without impacting application development cycles
    - Roll back on thread test failure

Summary

- Plan for maintenance and change for the entire mission lifecycle
  - Automation alone is insufficient
  - Missions need to adopt a culture and willingness for change
  - Automation and good process introduced in development and continued thru operations builds trust
- Containerization
  - Modularize system components for improved maintainability
    - Supports an agility in deployment approaches
    - Allows for isolation of functions
    - Manage change in smaller well defined objects
- Continuous Integration
    - Allow for automated change (e.g. patches, OS updates, address security vulnerabilities, infrastructure)
    - Reduce testing costs in operations

Questions?

Thank You!

The World's Forum for Aerospace Leadership